Using Logic Programming to model Multi-Agent Web Legal Systems – An Application Report

Paulo Quaresma and Irene Rodrigues
Departamento de Informática
Universidade de Évora
Évora, Portugal
pq,ipr@di.uevora.pt

ABSTRACT

A logic programming framework for the definition of cooperative multi-agent legal web information retrieval systems is proposed. Cooperation is achieved through the use of dialogue processing techniques, namely, the inference of the user intentions and the existence of a pro-active system behaviour, which tries to help users in their searches.

The proposed architecture has a core IR module, which accesses the legal knowledge bases, and three specialised logic programming agents: an agent manager that receives the user web initial requests and it establishes the connection with a specific user agent; a user agent, which is specific to each user, and it has information about the user profile and the previous interrogation context; and an agent monitor that informs the agent manager of the latest changes in the knowledge bases allowing these changes to be transmitted to all users which have one of their previous queries results changed.

The logic programming modules were defined using dynamic logic programming and LUPS, a language for updates [1, 3]. The proposed framework was implemented in a Linux environment using XSB Prolog and it was applied to the legal knowledge base of the Portuguese Attorney General [15].

The evaluation results show that the integration of dialogue processing techniques with a legal IR system, allow an improvement of the system, namely, decreasing the average number of interactions per query.

1. INTRODUCTION

Legal information retrieval systems are rapidly increasing their complexity, namely, the size and the content of their knowledge bases. As a consequence much work has been done trying to develop more powerful legal IR systems, integrating reasoning mechanisms with the IR modules, and creating third generation knowledge based systems [19]. However, there has not been much work trying to incorporate human-computer interaction techniques into legal IR systems. In fact, legal IR systems need a top dialogue layer, giving them the capability to interact cooperatively with the users. Cooperation can be achieved through the inference of the user intentions and goals, the pro-active organization of the set of retrieved documents accordingly with the users goals, and the interpretation of the queries taking into account the previous interaction context.

We propose a new system, which integrates dialogue processing techniques with a first generation legal IR system, and that is able to achieve a better degree of cooperativeness and to reduce the average number of interactions needed to retrieve the intended set of documents. In fact, the main goals of this work are to show the need for dialogue capabilities in legal IR systems and to propose a framework able to handle these requirements.

As a methodology for developing our system, we decided to create specialized legal agents, which would communicate between them, with the users, and with the IR knowledge bases. These legal agents need to be integrated in a general architecture and they should incorporate several typical characteristics from the agents theory: rationality, memory, cooperativity.

As logic programming has been previously used with success to model rational agents and their interactions [12, 3], we propose the use of the logic programming paradigm to model legal agents that behave in a rational and cooperatively way. Namely, we would like our legal agents to be able:

- To infer the "real" user intentions. When a user asks for documents with a particular keyword, usually he is interested in documents that may not have that keyword and he is not interested in all documents with that keyword [10, 13].
- To establish cooperative interactions with the users. The system, taking into account the inferred user intentions, should be able to help the user in the process of refining his queries.

In order to achieve these goals the logic legal agents need

the capability:

- To model the users mental state (beliefs, intentions, goals)
- To record the previous user interactions with the system (user questions and the system answers) [5, 6].
- To infer new user attitudes from their mental state, the user interaction history, and the user actions.
- To plan the system's actions, using the inferred user attitudes and the system's cooperative rules of behaviour.
- To interact with the information retrieval module, in order to obtain answer sets of relevant documents;
- To obtain new partitions (labelled clusters) of the retrieved set of documents, taking into account the user model [17, 7].

On the other hand, an architecture for a web legal information retrieval system must take into account that:

- The number of users registered in the system is large (thousands).
- Typically there are more then one user using the system at the same instant.
- The users may interrupt their session for large periods of time.
- The users would like to be informed when a previous query result changes due to updates in the knowledge bases.

Taking into account the described requisites and constraints, the proposed architecture is based on three specialised logic programming agents:

- An agent manager that receives the user web initial requests and it is responsible to establish a connection with a specific user agent;

  This agent receives the user initial requests from his web browser; it verifies if the user is registered and if it has no pending requests; then, if necessary, it launches other agent, the user agent.

- A specialised user agent that given an user request and its interrogation context is able to cooperatively interact with the user (inferring the user intentions, and planning the system cooperative actions).

  The agent process when it is launched consults the user database (profile and previous context); receives the user requests, infers the user attitudes, plans its actions, accesses the knowledge bases, answers the user; and updates the user database (interrogation context).

- An agent monitor that informs the agent manager of the latest changes in the knowledge bases. The changes are transmitted to all users which have one of their previous queries results changed.

  This agent consults all the users database to check for differences in the user query results. When there are changes, the agent sends a request to the user agent to inform the user about these changes.

This 3-agent architecture was applied to the Portuguese Attorney General information retrieval system [15] and it is available in the web (http://www.pgr.pt – in Portuguese).

Our architecture is, as far as we know, different from the other legal IR systems available. In fact, the integration of the dialogue processing knowledge (user intentions and goals, interaction structure, actions and behaviour rules) with a legal IR system, is a new approach and it allows us to model nicely the system's behaviour (cooperative, proactive). Moreover, the use of a multi-agent architecture allowed us to modularize our solution, specializing the agents, and being able to handle future extensions of the system (through the use of new agents or the extension of existent ones).

Some of the modules of our system can be compared with other existent legal IR systems. For instance, our IR module is based on SINO [8] and it was changed in a way that has many similarities with the work described in [4], namely, allowing the extraction of textual information using localization, inference, and controlled vocabulary. As in [11], we are also able to use concepts and a concept taxonomy in order to retrieve sets of documents. On the other hand, in the dialogue processing domain, our approach has similarities with the work of Carberry [5] and Litman [9] through the use of speech acts to recognize plans and with Pollack[13], which represents plans as mental attitudes.

2. DYNAMIC LP AND LUPS

Before describing the systems' architecture it is necessary to briefly present the dynamic logic programming paradigm and the language used to represent actions[1].

2.1 Dynamic Knowledge Representation

Given an original knowledge base KB, and a set of update rules represented by the updating knowledge base KB', it is possible to obtain a new updated knowledge base $KB^* = KB \oplus KB'$ that constitutes the update of the knowledge base KB by the knowledge base KB'. In order to make the meaning of the updated knowledge base $KB \oplus KB'$ declaratively clear and easily verifiable, in [1] there is a complete semantic characterisation of the updated knowledge base $KB \oplus KB'$. It is defined by means of a simple, linear-time transformation of knowledge bases KB and KB' into a normal logic program written in a meta-language. As a result, not only the update transformation can be accomplished very efficiently, but also query answering in $KB \oplus KB'$ is reduced to query answering about normal logic programs.

[1]This section is based on a previous work describing LP and LUPS [3]

2.2 Language for Dynamic Representation of Knowledge

Knowledge evolves from one knowledge state to another as a result of knowledge updates. Given the current knowledge state KS, its successor knowledge state $KS' = KS[KB]$ is generated as a result of the occurrence of a non-empty set of simultaneous (parallel) updates, represented by the updating knowledge base KB.

Dynamic knowledge updates, as described above, did not provide any language for specifying changes of knowledge states. Accordingly, in [2] it was introduced a fully declarative, high-level language for knowledge updates called LUPS "Language of UPdateS") that describes transitions between consecutive knowledge states KS_n. It consists of update commands, which specify what updates should be applied to any given knowledge state KS_n in order to obtain the next knowledge state KS_{n+1}. In this way, update commands allow us to implicitly determine the updating knowledge base KB_{n+1}. The language LUPS can therefore be viewed as a language for dynamic knowledge representation.

The simplest update command consists of adding a rule to the current knowledge state and has the form: $assert\ (L \leftarrow L_1, \ldots, L_k)$. For example, when a law stating that abortion is illegal is adopted, the knowledge state might be updated via the command: $assert\ (illegal \leftarrow abortion)$.

In general, the addition of a rule to a knowledge state may depend upon some preconditions being true in the current state. To allow for that, the assert command in LUPS has a more general form:

$$assert\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \quad (1)$$

The meaning of this assert command is that if the preconditions $L_{k+1}, \ldots, L_m$ are true in the current knowledge state, then the rule $L \leftarrow L_1, \ldots, L_k$ should hold true in the successor knowledge state. Normally, the so added rules are *inertial,* i.e., they remain in force from then on by inertia, until possibly defeated by some future update or until retracted.

However, in some cases the persistence of rules by inertia should not be assumed. Take, for instance, the simple action *request help*. This is likely to be a one-time event that should not persist by inertia after the successor state. Accordingly, the assert command allows for the keyword *event*, indicating that the added *rule* is *non-inertial.*

$$assert\ event\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \quad (2)$$

Update commands themselves (rather than the rules they assert) may either be one-time, non-persistent update commands or they may remain in force until cancelled. In order to specify such *persistent update commands* (which we call *update laws*) there is the syntax:

$$always\ [event]\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \quad (3)$$

To cancel persistent update commands, we use:

$$cancel\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \quad (4)$$

To deal with rule deletion, we employ the *retraction* update command:

$$retract\ (L \leftarrow L_1, \ldots, L_k)\ when\ (L_{k+1}, \ldots, L_m) \quad (5)$$

meaning that, subject to precondition $L_{k+1}, \ldots, L_m$, the rule $L \leftarrow L_1, \ldots, L_k$ is retracted. Note that cancellation of a persistent update command is very different from retraction of a rule. Cancelling a persistent update means that the given update command will no longer continue to be applied, but it does not remove any inertial effects of the rules possibly asserted by its previous application(s).

3. THE SYSTEM ARCHITECTURE

As it was pointed out in section 1 the architecture is based on an information retrieval module and three specialised logic agents:

- An information retrieval module, which is the responsible for accessing the legal knowledge bases and interacting with the other legal logic agents.
- An agent manager that receives the user web initial requests and it is responsible to establish a connection with a specific user agent;
- A specialised user agent that given an user request and its interrogation context is able to cooperatively interact with the user (inferring the user intentions, and planning the system cooperative actions).
- An agent monitor that informs the agent manager of the latest changes in the knowledge bases. The changes are transmitted to all users that have one of their previous queries results changed.

As it can be seen in figure 1, each user initially communicates with the agent manager, which redirects the event to the specific user agent (launching the user process, if needed). In order to obtain a cooperative answer, each user agent infers the user intentions and it interacts with the information retrieval module. Afterwards, it updates the interaction structure and communicates the answer to the user.

On the other hand, the monitor agent is always accessing the databases and the interaction structures trying to detect changes in the given answers. Then, it informs the agent manager, which will inform the change to the user process agent.

This architecture was implemented in a Linux environment using XSB Prolog.

4. THE INFORMATION RETRIEVAL

The information retrieval system is based on SINO [8] from the AustLII Institute. SINO was changed in order to be adapted to the Portuguese Language. Namely, the new system uses the Portuguese lexicon (more than 900,000 words) in order to handle morphological errors and to obtain the base queried word. For instance, if the user asks to be informed about documents where a specific word appear, the systems also searches for documents containing derived words (plurals for nouns, verbal forms for verbs, ...). We

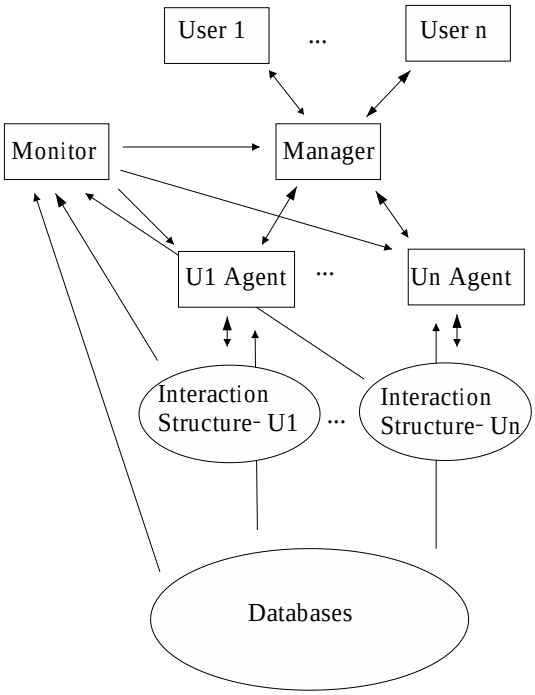


Figure 1: General architecture

also have a chart parser for a subset of the Portuguese language which is able to analyze the input and to extract a semantic interpretation of the sentence.

As a top layer over the basic IR system we are using a juridical terms thesaurus. This thesaurus is a result from another project: PGR - Selective access of documents from the Portuguese Attorney General.

The juridical terms thesaurus can be described as a taxonomy which has the relations:

- is equivalent to
  ex: law is equivalent to norm
- is generalised by
  ex: prime minister is generalised by minister
- is specified by
  ex: accident is specified by traffic accident
- is related with
  ex: desertion is related with traffic accident

The thesaurus is used to expand queries to include all the values that are equivalent or more specific or related, with the initial query. For instance, the user query "documents about accidents?" is expanded to the query "documents about accidents or traffic accidents or ...", which includes all the related and the more specific terms of accident.

The result is a core IR system, which has many similarities with the work described in [4], namely, allowing the extraction of textual information using localization, inference, and controlled vocabulary.

5. THE AGENT MANAGER

The agent manager receives the initial user requests, analyses them and redirects them to the respective user agent.

As it is not possible to have all the user agents running at the same time (we expect thousands of users), our solution was to have alive only the user agents correspondent to the active users. So, one of the agent manager tasks is to keep track of the active users and to launch the respective user agent, if needed.

This behaviour is described by the following logic programming rules:

always inform(Manager,AGENT_ID,A)
when request(USER_ID,Manager,A),
alive(AGENT_ID)

always alive(AGENT_ID) ← launch(AGENT_ID)

As described, these rules state that after a request from the user ID, the user process agent is launched (if needed) and the user action is redirected to it. Note that *launch* is a external system call, which will be the responsible to check if the AGENT_ID is running and, if needed, to start its process. As an example, suppose the agent manager receives a request from the user Joe to be informed about documents about militaries:

request(joe, manager, informref(manager, joe, X,
(doc(X), about(X,militaries)))).

Using the previous rules, the inference engine will try to satisfy $alive(joe\_agent)$ and it will execute the external function $launch(joe\_agent)$. This function will, eventually, start the *joe_agent* process. The effect of the main rule will be the action:

inform(manager,joe_agent,
request(joe, joe_agent, informref(joe_agent, joe,
X, (doc(X), about(X,militaries)))).

Note that the agent manager changes the actions' agents (manager to joe_agent) during the redirection of the actions.

Another task of the agent manager is to receive information from the monitor agent about database changes related to user queries (see section 7). The received information is redirected to the user agent and it can be used afterwards to inform the user of the database changes. The correspondent dynamic logic programming rule is:

always inform(Manager,AGENT_ID,P)
when inform(MONITOR,Manager,P)

For instance the following action:

inform(monitor, manager, changed(request(joe, ...)))

will be redirected to the user agent:

inform(manager,joe_agent,changed(request(joe,...)))

6. THE USER AGENT

This agent is specific to each user and processes its requests. It is launched by the agent manager to handle user requests. The agent receives the user request and it consults the user database to obtain the interpretation context for the user request. After, the agent needs to infer the user intentions and to plan its own actions. It accesses the knowledge bases, via the information retrieval module, and it answers the user.

In summary, to fulfill a user request the agent must:

- Load the user interaction context. The interaction context is a structure where all the previous user and system interactions are kept.
- Interpret the user act in the interaction context. The agent must infer the user intentions and beliefs from its actions (multimodal).
- Perform a set actions that are inferred, in order to fulfill the user intentions.
- Save the new interaction context for this user.

6.1 The user requests

This agent is expecting one of the following requests from the user:

- A request that results from a multimodal act, such as:
  - A request to return to a previous point of the interaction:
    $request(USER\_ID, AGENT\_ID, new\_context(Val))$
    The users have access to a representation of their interaction context, a tree with labels representing the requests. In order to generate this multimodal act the user clicks on the tree node representing the previous request. For instance, the user Joe may click on the tree node "Uranium" in figure 2 and the following event will be created:
    $request(joe, joe\_agent, new\_context(Uranium)$
  - A request to refine the previous selected set of documents with an expression (built with keywords connected by *and*s and *or*s):
    $request(USER\_ID, AGENT\_ID, refine(Expr))$
- A natural language expression in the form of a speech act (as in the example of the previous section):

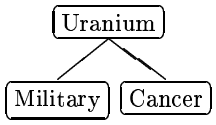


Figure 2: Interaction context tree

  1. $request(USER\_ID, AGENT\_ID, inform(AGENT\_ID, USER\_ID, S))$
  2. $request(USER\_ID, AGENT\_ID, inform\_ref(AGENT\_ID, USER\_ID, REF, S))$
- A request from the agent monitor to inform the user that one of his previous requests has a different result due to changes in the documents database.
  $request(Monitor, AGENT\_ID, inform(AGENT\_ID, USER\_ID, changes(Request))$

This set of requests will enable the inference of the user intentions and the planning of the agent's actions, as it will be shown in section 6.3.

6.2 The user interaction context

The user interaction context is kept in a dialogue structure. This structure records both user and system questions and answers. The structure is used to compute the meaning of a user query and to allow the user to return to a previous point of the dialogue and to build a new branch from there.

The Dialogue structure is made of segments that group sets of sentences (user and system sentences). The dialogue structure reflects the user intentions; it is built by taking into account the user and the system intentions. The dialogue segments have precise inheritance rules defining how segments heritage their attributes from the attributes of their sentences. The dialogue structure also enables the system to recognise and to generate pertinent discourse phenomena such as anaphoric references.

In order to interpret a user request this agent must insert the new request into the user dialogue structure and, as result of this process, a new structure is built.

The following LUPS rule controls the insertion of the user request in the dialogue structure:

always ds(DS_0) $\leftarrow$ update(U,DS_{-1},S,DS_0) *when* request(U,A,S), ds(DS_{-1}).

This rule means that, when there is a request S and a discourse structure DS_{-1}, the new structure is obtained from the update of the old structure with the new request.

To fulfill the user request, the agent will generate an action A, that is added to the old dialogue structure giving rise to a new one:

always ds(DS_1) $\leftarrow$ update(Ag,DS_0,A,DS_1) *when* action(Ag,A), ds(DS_0).

DS_1 will be the structure to be used for the interpretation of the next request by this user.

The *update* predicate defines how a new sentence is inserted in the existent discourse structure. As the code of this predicate is rather large, it will not be presented here. Instead, we will present the correspondent algorithm:

- If it is possible to join the semantic representation of the sentence with the last discourse segment, then the system assumes the user is refining his previous query. Suppose the user first asks about "militaries" and then asks about "cancer"; the system infers that he is interested in "militaries with cancer".
- It it is not possible to join the sentence with the previous discourse segment, the system tries to join it with a visible segment in the discourse structure (i.e. an ancestor of the last node in the tree dialogue structure).
- It it is not possible to join the sentence with any visible discourse segment, then the system assumes it is a sub-dialogue about a new subject and it creates a new segment without any ancestors in the tree dialogue structure.

6.3 The inference of user attitudes

In order to be collaborative our system needs to infer the user attitudes (intentions and beliefs). This task is also achieved through the use of logic programming framework rules and the dynamic LP semantics.

The system's mental state is represented by an extended logic program that can be decomposed in several modules (see [14] for a complete description of these modules):

— Description of the effects and the pre-conditions of the actions in terms of beliefs and intentions;

— Definition of behaviour rules that define how the attitudes are related and how they are transferred between the users and the system (cooperatively).

For instance, the rules which describe the effect of an inform and a request speech act from the point of view of the receptor (assuming cooperative agents) are:

always bel(A,bel(B,P))
when inform(B,A,P)

always bel(A,int(B,Action))
when request(B,A,Action)

In order to represent collaborative behaviour it is necessary to model how information is transferred from the different agents:

always bel(A,P) *when* bel(A,bel(B,P))

always int(A,Action) *when* bel(A,int(B,Action))

These two rules allow beliefs and intentions to be transferred between agents if they are not inconsistent with their previous mental state.

After each event (for instance a user question) received by the user agent, the agents' model (logic program) is updated with the description of the event that occurred. Then, the user agent calculates the new user model (well founded model of the logic program) and infers the user attitudes.

As an example, suppose the already presented example of a request from the user Joe: Documents about militaries?

This sentence is analyzed by a chart parser and a semantic interpretation module and it is translated into a *request* act to the agent manager:

request(joe, manager, informref(manager, joe, X,
(doc(X), about(X,militaries)))).

The agent manager informs the specialized user agent about this request:

inform(manager,joe_agent,
request(joe, joe_agent, informref(joe_agent, joe,
X, (doc(X), about(X,militaries)))).

Then, using the *request* rule, the user agent infers:

bel(joe_agent, int(joe,
informref(joe_agent, joe, X,
(doc(X), about(X,militaries)))).

Using the collaborative rule for the transference of beliefs, we have:

int(joe_agent,
informref(joe_agent, joe, X,
(doc(X), about(X,militaries)))).

Using the inferred intentions, the user agent will plan its actions and it will answer the user (as it will be shown in the next section).

6.4 The agent actions

The agent actions are the result of an abductive planning process, which tries to satisfy the inferred system intentions.

Suppose I is the set of inferred system intentions at a given time and that A is the set of correspondent intended actions (each intention has as object an action). The planning process is started by the creation of a new set of logical constraints, such that, for each intended action there exists an associated constraint stating that the action must be performed.

For instance, the intended action of the previous section will create the following constraint:

$false \leftarrow \neg informref(joe\_agent, joe, X,$
$(doc(X), about(X, militaries)))).$

This constraint states that if it is not possible to have the informref action performed, then we will have a contradictory state.

After the creation of this set of constraints, the user agent will abduce the set of actions needed to satisfy the constraints. Note that in a general domain, this task may not be always possible. However, in this domain we assume that the user agent is always able to satisfy the user intentions through the access of the IR modules (this means that it is always possible to answer a user query).

In our example, the solution is quite simple and it is the informref action:

informref(joe_agent, joe, X,
(doc(X), about(X,militaries))))

The abduced agent actions need to be performed and its results should be answered to the user. So, the actions are communicated to the IR module and its results are obtained. They may be of the following kind:

- A set of documents that match the user query.

  The set of documents is obtained by sending to the information retrieval search engine, SINO, the command "sino > search Q", with Q being the interpretation of the user request.

  As stated before, our information retrieval system is based on SINO, a text search engine from the AustLII Institute [8], that allows boolean and free text queries.

- A set of suggestions for further refinement of the user current query, i.e. a list of lists with keywords to be displayed.

  These suggestions are obtained by:

  - building labelled clusters of documents [16]. The clusters are obtained through the use of a Scatter/Gather algorithm [7] which clusters documents into topically-coherent groups based on a existent document classification (from the Portuguese Attorney General Office).
  - using domain knowledge. The domain knowledge will supply a set of concepts that can be used as suggestions for the user query refinement. Suppose we have the following rules:

    always military ← air_force
    always military ← navy
    always military ← army

    In this situation, if the user asks for "militaries", the user agent should be able to suggest the following possible refinements: air-force, navy, army. This behaviour is achieved through the calculation of the models that support the user goal and allowing the abduction of these predicates.

The following logic programming rule is used in order to obtain the described user agent behaviour:

always int(AGENT_ID,inform(AGENT_ID,USER_ID,R)
when abduced_actions(A), sino(A, R).

This rule means that, after having abduced a plan (i.e. a set of actions), the user agent sends the actions to the SINO module and it will receive an answer (as described before), This answer will be the object of a new intended inform action from the user agent to the agent. In our example, it might be the following intention:

int(joe_agent, inform(joe_agent, joe,
X in [doc1, doc32, ...],
refine_by(air_force,navy,army)))

6.5 The agent top goal

The agent top goal is to receive a request and to act cooperatively, namely, to infer the user intentions, plan the agent actions, obtain answers from the IR system and give them back to the user. Then, it saves its states and, if nothing happens after some pre-defined period of time, it launches the goal *terminates* (it assumes the user has closed the connection).

always act(SYSTEM,Actions)
when request(USER,SYSTEM,R),
infer_intentions(USER,I),
add_constraints(I),
plan_actions(P),
infer_intentions(SYSTEM,I),
obtain_actions(I,Actions).

always terminates
when act(SYSTEM,A), save_state, wait(T).

7. THE AGENT MONITOR

The agent monitor informs the agent manager about the users that must be informed of the latest changes in the documents database.

This agent runs after any change in the knowledge bases, and it tries to find the users that may have the results of their previous sessions changed by the update.

It consults all user databases to check for differences in each user query result. When there are changes the agent sends a new request to the agent manager process.

A user may define in his profile if wants to be informed when there are relevant changes in the documents database.

The main agent monitor tasks are:

1. To build a list with all the users that want to be warned whenever there are relevant changes in the documents database.

2. For each user in this list it opens the user database where the interaction context is kept and consults the dialogue structure.

3. Then, for each user request it checks if the new interpretation, after the changes, give the same results.

   This is done by requesting the user agent to check the old user query. In order to check if the request results are the same, the agent monitor must compare the new agent action with the old one (just the list of documents selected by the query).

4. Whenever there are changes in the results of a user request interpretation this agent sends a message to the agent manager with the request:

   request(Monitor,Manager,
   inform(Manager, USER_ID, changes(Request))

   The agent manager is responsible for sending this request into the agent user, which will use it in the next inference process.

8. INTERACTION EXAMPLE

Suppose the user (Joe) is a parent of a military that is currently in a peace mission in Kosovo. He is worried about his son possible health problems caused by uranium bombs and he wants to find out if there are similar cases in the legal knowledge bases.

[User - Q1:] Documents about militaries?

As it was already shown, this sentence is translated into the following speech act:

request(joe, manager, informref(manager, joe, X,
(doc(X), about(X,militaries)))).

This speech act is the basis of an inference process that supports the following system intention:

int(joe_agent, informref(joe_agent, joe, X,
(doc(X), about(X,militaries)))).

This intention is be used in the planning process and it leads to the abduction of the correspondent informref action. The abduced action is sent to the IR module and the answer is shown to the user:

[System - Q2:] There are 124 documents, listed in the right frame. For further refinement you may choose one (or none) option in each column:

Militaries belonging to the:
air-force
navy
army

Documents with keyword:
pensions
illnesses
...

Options in column 1 are obtained using rules that model legal knowledge about militaries. Options on column 2 are obtained by clustering the set of 124 documents.

[User - Q3:] chooses: army and illnesses.

This user action is interpreted in the context of the existent dialogue structure and it is inferred that it may be a refinement of the previous query:

request(joe, manager, informref(manager, joe, X,
(doc(X), about(X,militaries),
about(X,army), about(X,illnesses)))).

Then, the same approach is followed and the system answer is:

[System - Q4:] There are 13 documents, listed in the right frame. For further refinement you may choose one (or none) option in each column:

Documents with keyword:
pensions
cancer
...

Options in this column are obtained by re-clustering the 13 documents.

[User - Q5:] Documents about cancer.

Again, the user sentence is interpreted in the context of the dialogue structure and it is inferred that it might be another query refinement:

request(joe, manager, informref(manager, joe, X,
(doc(X), about(X,militaries),
about(X,army), about(X,illnesses),
about(X, cancer)))).

[System - Q6:] There are 5 documents that match the concept, I've listed them in the usual frame. For further refinement you may choose one (or none) option in each column:

Documents with keyword:
lungs
leukemia

[User - Q7:] Uranium?

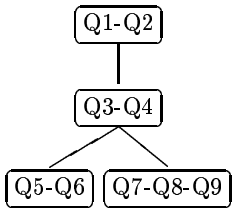
After this user query, the system infers that there are no documents about the conjunction of all the previous concepts so, probably, it is not a new refinement of the last query but a refinement of one of its ancestors (as described in section 6.2).

[System - Q8:] Since there are no documents of the 5 above that match your new constraint, I assume that you want your query to refine Q3, i.e. you are looking for documents about militaries and "army and illnesses" and "Uranium". Do you want to see these documents?

[User - Q9:] Yes. .....

This example shows some of the flexibility and cooperativeness of the system, allowing users to dynamically refine their queries and helping them in a pro-active way, giving hints and clustering the retrieved documents.

During the interaction, the tree representation of the dialogue is being inferred and displayed in a visual tree-diagram. As an example, the tree representation of the previous example would be the following:



9. EVALUATION

As it is widely known, the evaluation of legal systems with knowledge representation and reasoning capabilities are a very complex and difficult task [18].

The main goal of this evaluation process was not to evaluate the information retrieval module but the impact of cooperativity in the interaction process with the users.

We have defined two sets of users: one using the cooperative agent technology and the other accessing directly the information retrieval module. We have recorded their queries during the second semester of the year 2000 and the preliminary results seem to show that our system is able to help the users decreasing the average number of queries needed to obtain the desired documents (around 20%).

At the moment, we are beginning another evaluation process: users from both sets are asked to obtain specific documents and the number of iterations needed to retrieve the documents is measured. Finally, we will compare the results obtained with and without the cooperation module.

10. CONCLUSIONS

We have presented a logic programming based architecture for cooperative multi-agent legal web information retrieval systems. Cooperation is achieved through the integration of dialogue processing techniques with standard IR modules.

The architecture is based on three kind of agents: the agent manager, which interacts with the user and with the specialised user agents; the user agents, which have the knowledge specific to each user and interacts cooperatively with them; and the monitor agent, which tries to detect changes in the databases and to inform the users about them. The architecture was implemented using XSB Prolog over a legal information retrieval system (available in Portuguese from http://www.pgr.pt).

The evaluation results, showed that our cooperative system is able to help the users decreasing the average number of queries needed to obtain the desired documents (around 20%).

As future work, we intend to obtain more evaluation results (quantitative and qualitative) and to apply our system to other legal information retrieval systems, namely to non-Portuguese legal documents. We will also intend to apply the system to other domains.

Acknowledgements

We would like to thank the ICAIL referees for their helpful comments on the first version of this paper.

11. REFERENCES

[1] J. J. Alferes, J. Leite, L. M. Pereira, H. Przymusinska, and T. Przymuzinski. Dynamic logic programming. In *Proc. of KR'98*, 1998.

[2] J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Preliminary exploration on actions as updates. In M. C. Meo and M. Vilares-Ferro, editors, *Procs. of the 1999 Joint Conference on Declarative Programming (AGP'99)*, pages 259–271, L'Aquila, Italy, September 1999.

[3] J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Dynamic knowledge representation and its applications. In S. Cerri and D. Dochev, editors, *Proceedings of the 9th International Conference on Artificial Intelligence - Methodology, Systems, Applications (AIMSA'2000)*, number 1904 in Lecture Notes in Artificial Intelligence, pages 1–10, Varna, Bulgary, September 2000. Springer Verlag.

[4] T. Bueno, C. von Wangenheim, E. Mattos, H. Hoeschl, and R. Barcia. Jurisconsulto: Retrieval in jurisprudencial text bases using juridical terminology. In *Proceedings of the ICAIL'99 – 7th International Conference on Artificial Intelligence and Law*, pages 147–155. ACM, June 1999.

[5] S. Carberry and L. Lambert. A process model for recognizing communicative acts and modeling negotiation subdialogs. *Computational Linguistics*, 25(1), 1999.

[6] J. Chu-Carroll and S. Carberry. Response generation in planning dialogues. *Computational Linguistics*, 24(3), 1998.

[7] D. R. Cutting, J. O. P. D. R. Karger, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proc. 15th Annual Int'l ACM SIGIR Conf. on R&D in IR*, June 1992.

[8] G. Greenleaf, A. Mowbray, and G. King. Law on the net via austlii - 14 m hypertext links can't be right? In *In Information Online and On Disk'97 Conference, Sydney*, 1997.

[9] D. Litman and J. Allen. A plan recognition model for subdialogues in conversation. *Cognitive Science*, 11(1), 1987.

[10] K. E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 24(4), 1998.

[11] J. Osborn and L. Sterling. A judicial search tool using intelligent concept extraction. In *Proceedings of the ICAIL'99 – 7th International Conference on Artificial Intelligence and Law*, pages 173–181. ACM, June 1999.

[12] L. M. Pereira and P. Quaresma. Modelling agent interaction in logic programming. In O. Yoshie, editor, *INAP'98 - The 11th International Conference on Applications of Prolog*, pages 150–156, Tokyo, Japan, September 1998. Science University of Tokyo.

[13] M. Pollack. Plans as complex mental attitudes. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communications*. MIT Press Cambridge, 1990.

[14] P. Quaresma and J. G. Lopes. Unified logic programming approach to the abduction of plans and intentions in information-seeking dialogues. *Journal of Logic Programming*, 54, 1995.

[15] P. Quaresma and I. Rodrigues. Pgr: A cooperative legal ir system on the web. In G. Greenleaf and A. Mowbray, editors, *2nd AustLII Conference on Law and Internet*, Sydney, Australia, 1999. Invited paper.

[16] P. Quaresma and I. P. Rodrigues. Automatic classification and intelligent clustering for wwweb information retrieval systems. *Journal of Information Law and Technology (JILT)*, 2, 2000. http://elj.warwick.ac.uk/jilt/00-2/ – Extended and revised version of the BILETA'2000 paper.

[17] G. Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley, 1989. Reading, MA.

[18] A. Stranieri and J. Zeleznikow. The evaluation of legal knowledge base systems. In *Proceedings of the ICAIL'99 – 7th International Conference on Artificial Intelligence and Law*, pages 18–24. ACM, June 1999.

[19] J. Yearwood and A. Stranieri. The integration of retrieval, reasoning and drafting for refugee law: a third generation legal knowledge based system. In *Proceedings of the ICAIL'99 – 7th International Conference on Artificial Intelligence and Law*, pages 117–125. ACM, June 1999.